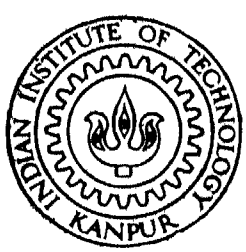
A LINUX based Asymmetric IP Router for Hybrid Networks

by
G MAHEEDHAR



DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

FEBRUARY 1998

A LINUX based Asymmetric IP Router for Hybrid Networks

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

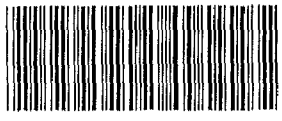
G Maheedhar

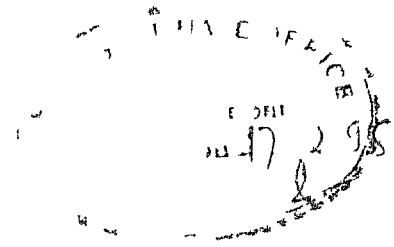
to the

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

February 1998





Certificate

This is to certify that the work contained in this thesis entitled *A LINUX based Asymmetric IP Router for Hybrid Networks* by G Maheedhar has been carried out under our supervision and that this work has not been submitted elsewhere for a degree

Sanjay K Bose
Department of Electrical Engineering
Indian Institute of Technology Kanpur

D Manjunath
Department of Electrical Engineering
Indian Institute of Technology Kanpur

Acknowledgements

I take this opportunity to thank Dr S K Bose and Dr D Manjunath for the confidence they showed in me and for providing me invaluable guidance throughout this thesis work I am grateful to Mr Amitabha Roy for the patience he showed in teaching me the basics of the network implementation His suggestions and guidance helped me through the tough times of studying the LINUX source code This work would not been possible without his valuable guidance I thank Mr Vivek Mudgil for helping me in setting up the test bed

I would like to express my gratitude to all in the ERNET and Telematics Labs with special mention to Sandhya Madam Neeru Madam T Srinivasa Rao Mr Bhatnagar Vineet Ila Madhuri

My sincere thanks to Ritvik for helping me in solving most of my problems during my thesis work I would thank my friends Hema Bhaskar Biplab Sonia Hemchandra Murty Vivek Sudheer for their encouragement and suggestions

Finally I would like to thank Sanjay Bose and his family for making my stay at IIT Kanpur memorable

G Maheedhar

Abstract

This thesis aims at the development of a cost effective high speed data transmission system Most internet traffic is asymmetric in nature Usually a user machine accessing the internet will send very little data in the forward direction but will tend to download much larger volumes of data in the reverse direction In case the forward and return paths can be split with high speed links provided in the reverse direction while using the much lower speed links in the forward direction significant improvements in the overall download rates may be obtained A hybrid network of this type using a combination of two or more communication channels will provide high speed internet access A PC Linux platform has been used to develop an asymmetric router for use in such a system This thesis presents the design and implementation of this router and studies its performance in a test set up designed by us

Contents

List of Figures

Chapter 1

Introduction

A common way for home users or small organizations to access the internet is with a modem which uses the telephone lines to connect to an Internet Service Provider (ISP) The computer dials up the ISP for access through its modem It is then connected via the modem to the ISP which is in turn connected to the internet with a high speed link The Internet Service Provider will charge the user for the connection either on a flat basis or on a connection time basis If the user s machine has a valid internet (IP) address then this may be directly used for providing the required internet services In addition the ISP may also dynamically provide the user with a valid IP address for his/her machine which will be assigned from a range of addresses available to the ISP this assignment will be a temporary one which will provide a valid identification for the user s machine only for the duration of the connection

The user organization may also want to connect a number of machines on its internal network to the internet through its ISP This is comparatively more complicated as each machine on the user s internal network may seek to establish one or more separate connections to different hosts on the internet One way of achieving this is through a masquerading process where all the machines on the internal network effectively hide behind a special masquerading host This masquerading host is the one which actually connects to the ISP The machines on the internal network appear as a single IP address to the outside world this is the IP address of the masquerading

host which in turn may be the one dynamically assigned to it by the ISP The masquerading host will also be required to transparently maintain and manage all the connections that the internal machines establish with other hosts on the internet Typical internet access is shown in Figure 1 1

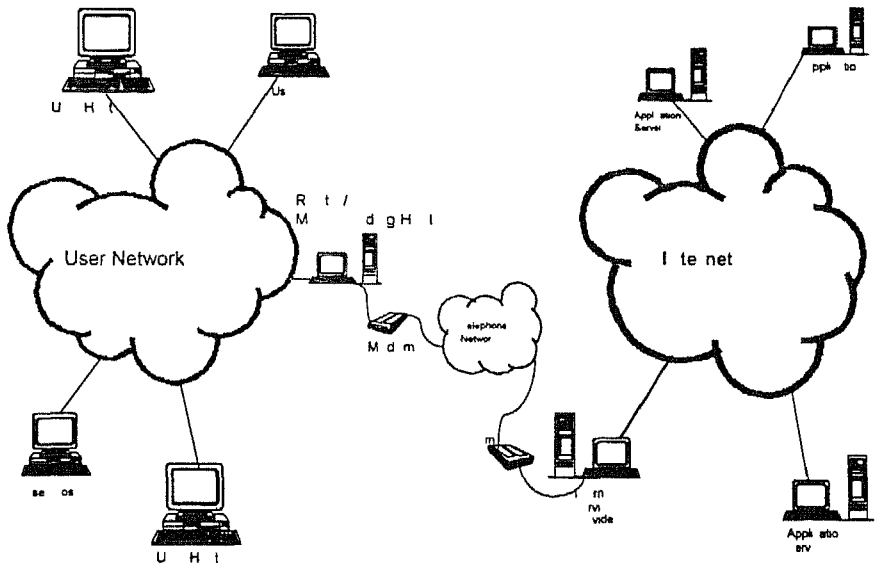


Figure 1 1 Typical Internet Access

In typical situations normal telephone lines will be used for accessing the ISP This can provide very limited data transfer rates between the user s machine and the internet This bottle neck occurs because of the relatively low speed of the telephone lines connecting the user s machine and the ISP where these lines are used to transfer data both from the user to the ISP and vice versa (The ISP will typically have a much higher speed connection to the internet) This will limit the speed with which a user can download information from the internet This bottle neck hurts most in applications such as file transfers (FTP) or world wide web (WWW) access as here the data transfer is highly asymmetric in most commonly encountered situations typically there would be much more data going from the internet/ISP to the user than in the other direction For such applications significant performance improvement in data transfer rates may be easily obtained if a high bandwidth channel can be made available for carrying data from the internet to the user This will require a special purpose router on the user s premises and an appropriate application server (possibly

co located with the ISP or placed at some other suitable location) configured to handle such an asymmetric connection With this arrangement the effective download speed may be increased to levels which would be virtually limited only by the speed of the high bandwidth return channel to the user while still using the low speed telephone channels for carrying data from the user to its ISP

In principle any high bandwidth channel may be used for this purpose if it can provide data transfer from the designated application server to the user However using satellite channels for this purpose is an extremely attractive idea as these channels not only provide very high bandwidth but also have a very wide geographical reach The satellite will be able to provide this high speed return channel service to any user within its foot print as long as the user can install a receive only satellite dish to receive data from it (This dish may be shared with normal direct to home satellite reception unit which would bring down its establishment and maintenance costs even further) This receive only satellite dish is small in size and would be cheap to manufacture install and maintain High speed cost effective connectivity to the internet will then be available to the user by using a modem for uplink traffic to its ISP and a receive only satellite dish for downlink traffic from a suitable internet application server

This convergence of satellite and terrestrial technologies has given rise to Hybrid Networks providing home users and small organizations with high data rates in a cost effective way for downloading information from the internet In this context the term hybrid refers to the fact that the network uses a mix of satellite and terrestrial communication links These hybrid networks may also merge satellite and wireless systems with the internet and with other cellular cable and telephone networks Such a hybrid networks differs from ordinary networks in that there will generally be a large mismatch in the bandwidths available in the two directions of data transfer In addition there may also be a relatively large transmission delay in one path i e the satellite channel will have a one hop delay of about 0 27 seconds as opposed to the almost instantaneous response of a terrestrial channel In effect this means that the delay bandwidth product of the path in one direction will be significantly higher than

in the other direction This will require special handling at various layers of the network This problem is especially important as the hosts on the internet will not be aware of the special asymmetric connection to the user and should be accessed as they would normally be by a usual terrestrial user or network

For the work reported in this thesis we assume that the hybrid network should operate so that all the machines on the entire client network at the user s premises may access the internet in an asymmetric way The client network is placed behind a masquerading host The client machines in the network dial out to the ISP via the masquerading host to connect to the internet The return data is received via a satellite dish by the masquerading host and is routed to the respective client machines

A commercial version of such an application is provided by DirecPC [1] which may be used to connect one client machine to the internet for a maximum download throughput of 400 Kbps Once DirecPc is installed the data from the internet is received by the user through a receive only satellite dish and the upstream data from the user is sent out using telephone lines and the ISP connection DirecPc installation comprises of loading a special driver to the client machine We have extended this idea to one where an client network is to be provided such internet access through a special router which has been developed by us This allows us to do service specific routing where we can selectively choose the types of services (e g WWW FTP) for which the asymmetric hybrid network will be used the other network services/packets (e g TELNET ICMP) will use the normal terrestrial network connection through the ISP A companion thesis [2] has designed the special internet application server which will get data from the internet to be sent to the user through the satellite broadcast channel (This special server is not necessary as the router described here can be used even with a proxy server set up for this purpose) The client machines in the client s network and the internet hosts operate completely transparently expect that they enjoy high download data rates The client machines can operate using any appropriate operating system and only need to keep their default route to our special router which also operates as the masquerading host The latter then takes the responsibility of receiving data via the satellite dish and sending

it out through the ISP for the services for which high speed asymmetric access is to be used for the other services the router/masquerading host will use the ISP connection through the telephone lines for data transfers in both directions

The subsequent chapters of the thesis are organised as follows In Chapter 2 we describe the hybrid network and the problems encountered in operating with such a network The methods used by us to over come these problems have been discussed here An algorithm developed by us for dropping acknowledgments when the low band width return path becomes congested has also been presented Chapter 3 describes the design and implementation of various sub systems required by our design These are based on the LINUX operating system which is freely available We have used this freeware to implement our design Chapter 4 describes the configuration of various systems in the hybrid network The configuration files used the routing entries that are made and other options used have been explained The performance of our router has been tested in a experimental test set up This set up and the results of these tests have been described The summary conclusions of the thesis and some suggestions for further work are presented in Chapter 5

Chapter 2

Hybrid Network

Hybrid networks merge multiple communication technologies and are emerging as the most viable solution for the ever increasing demand for bandwidth and data services With the increase in the number of networks and as more and more users access the web there is an exponential growth in bandwidth demand that cannot be satisfied with the existing internet capacity and traditional data services Typically a common internet surfer who connects from home will be frustrated by the speed at which his/her service is working This is because of the low data rate of the classical dial up connection set up via an Internet Service Provider Typical Internet access of a common user is asymmetric with low data traffic from the user host to the internet application server and high data traffic in the opposite direction Even in many communication applications there is a need to transmit much information primarily in one direction between two points and much less information in the opposite direction This is typically the case in file transfer and database services In situations where there is an asymmetry in traffic we can make the transmission bandwidth also asymmetric so that we can optimize the resources of the network A one way high bandwidth channel (such as those provided by a broadcast satellite link) may be used for the bulk information transfer and a parallel terrestrial link for the low bandwidth portion of the application traffic This is the motivation for the evolution of hybrid networks In a hybrid network (parallel satellite and terrestrial channels) the satellite receive dish need not have a transmit capability it can therefore be a much less

expensive receive only terminal In addition to cost savings it is possible that improvement in network performance may also be achieved since the terrestrially carried traffic will not suffer high propagation delay incurred through satellite links We can view such a hybrid network as consisting of a low delay terrestrial sub network and a high bandwidth and possibly a high delay sub network This hybrid network can provide *cost effective high speed data transmission* provided the asymmetry of the hybrid network is accounted for in the way access is provided to the user An example of such a hybrid network using a combination of terrestrial links through a standard ISP and a broadcast satellite link has been shown in Figure2 1 In the following description we have used the network configuration shown in this figure it should however be understood that the broadcast satellite link may be substituted by any appropriate high speed connection with either high or low transmission delays Our proposed design will be applicable to all such situations

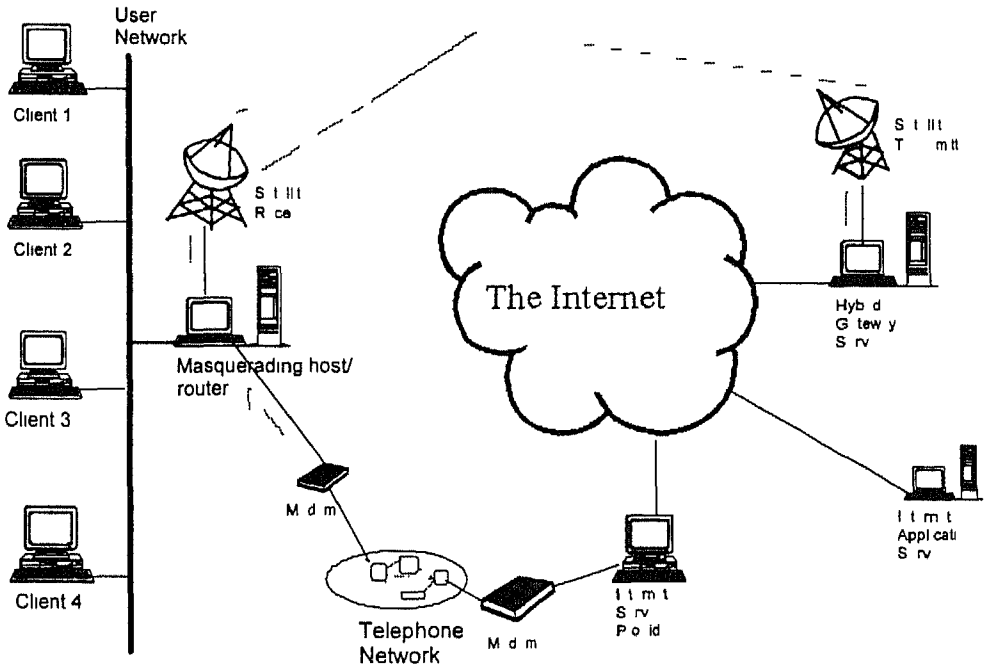


Figure 2 1 Asymmetric Internet Access in Hybrid Network

As shown in Figure 2 1 the client machines on the user network have access to the to the Internet via the masquerading host/router The functionality of the masquerading host is to represent the complete user network as a single host This means that all the packets going outside from the user net to the internet appear as though they have originated from the masquerading host Since the complete user network is hidden behind the masquerading host there is no need for the user machines to get officially assigned IP addresses There are three blocks of private IP numbers set aside for assigning to the user machines on any non connected network ([5] can be referred for detailed information) A chunk of addresses from any of these three blocks can be used in the local net The masquerading host has to be assigned a valid IP address which would normally be provided by the Internet Service Provider Depending on the Internet Service Provider s usage policy this IP address may either be dynamically assigned or may be assigned permanently

The masquerading host has three network interfaces One interface is connected to the user network The second interface is connected to bi directional telephone line to dial out into the Internet Service Provider for accessing the internet The third interface is connected to the receive only satellite dish The IP address of the High Speed Down Link (HSDL) interface is to be assigned by the Satellite Transmission Provider The traffic from the user network to the Internet goes via the telephone lines and the return traffic comes via the receive only satellite dish For achieving this functionality the hybrid gateway server is used All the requests and data sent by the client machines are to be sent to the hybrid gateway This is accomplished by using tunneling or IP within IP encapsulation The hybrid gateway server in turn gets the data from the internet application server and sends the data to the satellite transmitter This data is received by the masquerading host (asymmetric router) via the HSDL interface We will henceforth refer to the masquerading host as the asymmetric router as it routes the traffic in an asymmetric way the implementation of this using a Linux based machine is the topic of this thesis The data received by the asymmetric router is demasqueraded and is sent to the appropriate client machine on the user network All the acknowledgments generated by the client machines are sent to the internet

application server via the telephone lines The system is so developed that all the client machines and internet hosts are transparent to these changes and the routes that the packets take This design will allow the Hybrid System to work with all existing TCP/IP networks any Internet Service Provider and will work with any host or router anywhere in the system

In this setup only the asymmetric router and the hybrid gateway server do the specific tasks required for achieving the desired asymmetric data transfer The main function of the asymmetric router is to masquerade all the packets coming from the local network via the telephone line interface but to represent to the external world as though that they have been masqueraded out of the HSGL interface So the hybrid gateway server routes the packets to the high speed sub network The main function of the hybrid gateway server is to get the data from the internet application server and send it to the satellite transmitter The present TCP/IP protocols are designed to obtain high throughput With the asymmetry in the transmission of data and the merging of multiple transmission modes of different characteristics the throughput of the TCP/IP protocols has to be reviewed so that the hybrid network can actually attain this high throughput objective

2 1 Performance bottlenecks

If a satellite broadcast channel is used then a one hop transmission delay of 270 milliseconds is introduced in the return path other media have higher or lower transmission delays depending on the nature of the media and its implementation for data communications This delay increases the round trip time on which the flow control of the TCP protocol will depend The hybrid system is developed so that it can work with the existing TCP/IP protocols To provide reliable delivery TCP protocol uses end to end flow congestion and error control mechanisms A large number of acknowledgments will also be generated if a large amount of data is being downloaded These acknowledgments may cause congestion the low bandwidth path and may restrict the overall throughput of the Hybrid Network

2 1 1 High delay in the return path

The flow control mechanism in TCP protocol depends on the window size and the round trip time (RTT) The satellite transmission in the return path of the data introduces a one hop transmission delay of 270 milliseconds Therefore the RTT will increase significantly compared to the RTT in a completely terrestrial system As a result of this the sender on the internet will have to wait for more time to receive the acknowledgments that it expects To overcome this problem the window size on the client machine can be increased The Hybrid Network is designed so that the client machines can work on any platform using any Operating System or TCP/IP stack Therefore changing the window size (typical default window size is 4096 bytes) on the client machines will violate our design philosophy of complete transparency This problem is solved by effectively using the hybrid gateway server All the data that is processed (received and transmitted) by the asymmetric router has to pass through the hybrid gateway server The hybrid gateway server is made to do some intelligent processing of the data so that the effect of the large RTT may be appropriately handled

2 1 2 Congestion in the low bandwidth path

The down link through the satellite channel or through a high speed link will have a very high rate i e typically in the order of several megabits per second When a data file of reasonable size is being transferred a large number of acknowledgments will be generated These are to be sent on the low bandwidth telephone lines This would cause congestion in the forward path which will then delay the acknowledgments from reaching the application hosts As the result the receive windows on the internet application hosts will also not be suitably updated This will tend to reduce the over all throughput of the system unless something is done to combat this problem

2 2 Solutions

We have taken care of the large RTT effect by using TCP spoofing at the hybrid gateway server An algorithm is proposed to do acknowledgment dropping for reducing congestion in the low bandwidth path Special provisions have also been made for interactive applications which are more sensitive to the transmission delay

2 2 1 TCP spoofing

The bandwidth delay problem is solved by transparently splitting the end to end TCP connections into two parts the conventional terrestrial portion and the hybrid portion The connection splitting is shown in the Figure 2 2 The hybrid gateway sever connects the two networks (Hybrid Network and the The Internet) The connection between each client machine on the user network and the corresponding internet application server is split into two different connections i e one connection between the client machine to the hybrid gateway server and the second connection between the hybrid gateway server and the internet application server (In our implementation we used the Squid Proxy Server software to do this connection splitting) The hybrid gateway server has a large buffer and can therefore advertise a large transmit window to the asymmetric router (client machines) and a large receive window to the internet application server The hybrid gateway server acknowledges the internet application server on the behalf of the client machine i e it receives and buffers all the data from the internet application server keeps these in its buffer This data is sent to the satellite transmitter When the client machines receive the data they send their acknowledgments to the hybrid gateway server The hybrid gateway server then removes the data after it receives the corresponding acknowledgment from the client machine

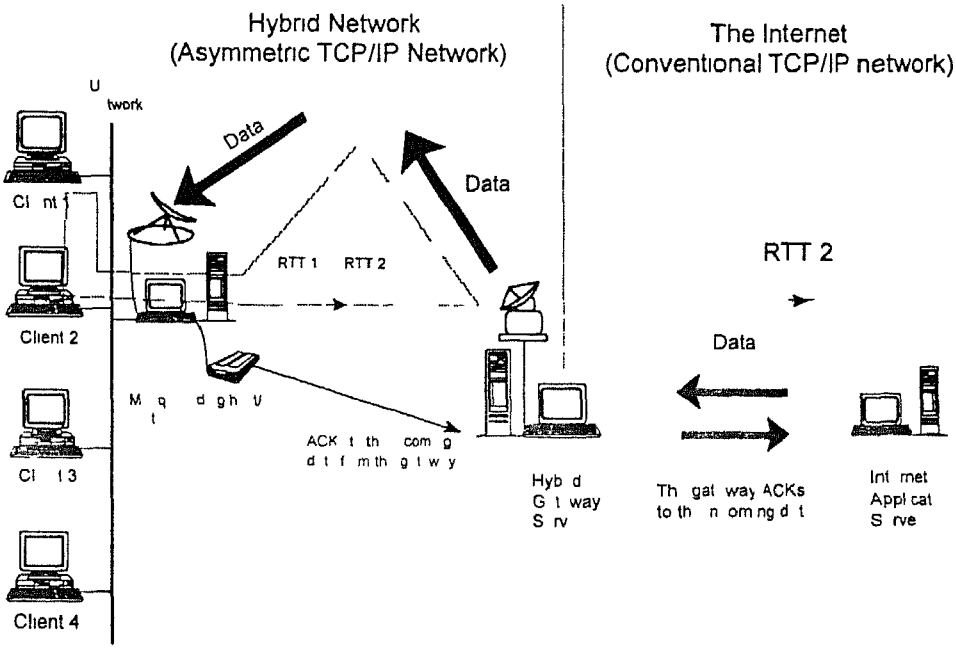


Figure 2 2 Splitting of TCP connection

Using this approach we have successfully isolated the Hybrid Network from the internet as shown in Figure2 2 The hybrid gateway server will try to get the data from the internet application server as fast as it can and send acknowledgments for it to the latter It will also send the data to the client machine Another advantage of TCP spoofing (i e the splitting of the connection) is that the RTT in the Hybrid Network will be nearly constant Since there is nearly no variation in the RTT the round trip delay estimation of the client machines will more effective irrespective of the delay even when the delay is large

2 2 2 Acknowledgment dropping

The large data transfer via the HSDL interface generates acknowledgments which are sent back to their respective application server via the low bandwidth telephone lines As the download data rate is large a large amount of acknowledgments are generated which may cause congestion in the return path We have developed an algorithm to drop the pending acknowledgments so that the congestion is reduced This is implemented at the asymmetric router The algorithm and its implementation is given in Section (3 3 2) Note that the reason we can drop the acknowledgments

without any adverse effect is because TCP uses a cumulative acknowledgment scheme i e it acknowledges for bytes instead of packets In our approach we have implemented this by only sending the most recent ACK for a particular connection dropping the other pending ACKs for this connection Proper care has however been taken to ensure that piggy backed acknowledgments are not dropped

2 2 3 Selective Asymmetric Transmission

Interactive applications like telnet (where users constantly input data) and applications which expect small size responses (such as domain name queries) should avoid using the high bandwidth channel if it also has a high delay e g a satellite channel This is because the high delay will make the users feel that the link is slow We have designed our asymmetric router such that there is a provision for the system administrator to select the return path depending on the type of service i e either via the telephone or via the satellite The system administrator can decide for which services the hybrid transmission is required and set the filter rules by using the TCP port numbers Generally for applications that require large data downloads the satellite link is used as the return path For applications which are sensitive to the network delay the modem connection will be used as the return path

The complete firewall facilities can be used by the system administrator of the asymmetric router to impose access restrictions on the hosts of the user network Firewall rules can be setup to restrict the access based on services destination networks a particular domain etc

Chapter 3
System Description

We used the Linux Operating System on a PC platform for the implementation of the asymmetric router We implemented our design on the TCP/IP protocol suite The internals of the networking software of the Linux Kernel for the routing functionality and masquerading were studied The IP packet filtering rules are used to make the selective asymmetric transmission We designed our system such that the congestion on the low bandwidth link is reduced The networking software was modified so that the system can be used as the asymmetric router in a Hybrid Network environment A brief introduction and overview of TCP/IP is given Details of the networking support utilities modules internals and various issues of the Linux Kernel are discussed This chapter concludes with the design and implementation details of the asymmetric router

3 1 Overview of TCP/IP Protocols

TCP/IP is a set of protocols developed to allow cooperating computers to share resources across a network The protocol suite was developed by a community of researchers centered around the ARPAnet which evolved as the best known TCP/IP network Famously known as The Internet this presently has thousands of networks of all kinds connected with routers with TCP/IP as the communication protocol TCP/IP is a large and complex protocol and it is not possible to summarize all its details in one section Comprehensive descriptions on the design and implementation of TCP/IP are given in [4] [7] and should be consulted for specific details Some of the major issues of interest to our design effort are summarized in the following section

TCP/IP is a basket of protocols which work in coordination for providing connectionless reliable data transfer The term connectionless refers to the fact that in the transfer of data packets between two computers each packet is treated independently from all others A sequence of packets sent from one computer to another may travel over different paths The transfer is reliable in the sense that each received packet is acknowledged and hence a lost packet will be resent by the sender until it gets the corresponding acknowledgment The basic protocols in the TCP/IP suite are the IP TCP UDP and ICMP protocols

Various application protocols are developed on top of these protocols for doing specific tasks e g transferring files between computers (FTP) sending mail (MAIL) or finding out who is logged in on another computer (RWHO) Any real application such as Mail will use several of these protocols First there is a protocol for mail This protocol defines a set of commands which one machine sends to another e g commands to specify who the sender of the message is who it is being sent to and then the text of the message Mail does not take care of error control or reliability issues and simply assumes that that there is a way to communicate reliably between two computers Mail like other application protocols simply defines a set of commands and messages to be sent It is designed to be used together with TCP and IP TCP is responsible for making sure that the commands get through to the other end It keeps track of what is sent and retransmits anything that did not get through If any message is too large for one packet e g the text of the mail TCP will split it up into several packets and make sure that they all arrive correctly These functions of TCP are needed for many applications and they are put together into a separate protocol rather than being part of the specifications for sending mail TCP is a library of routines that applications can use when they need reliable network communication with another computer TCP calls on the services of IP for the delivery of the packets to the destination In order to perform its functions TCP may use the IP protocol IP is a library of routines that TCP calls on but which is also available to applications that do not use TCP

The strategy of building several levels of protocols is called *layering* We think of the applications programs such as mail TCP and IP as being separate layers each of which calls on the services of the layer below it TCP/IP is a four layered architecture The application protocol such as mail will be the topmost layer The next layer is the TCP layer which provides reliable communication services needed by the applications This is followed by the IP layer which provides the basic service of getting packets to their destination The lowest layer is the physical layer specifying the protocols needed to manage a specific physical medium such as Ethernet or a point to point line

TCP/IP assumes that there are a large number of independent networks connected together by routers The user should be able to access computers or other resources on any of these networks Packets will often pass through several different networks before getting to their final destination The routing needed to accomplish this should be completely invisible to the user As far as the user is concerned all he or she needs to know in order to access another system is its internet address

TCP transfers data in packets where the individual packets are sent through the network There are provisions to open connections to systems TCP determines the size of data in the packets and sends these to the other end At the destination these packets will be recombined into the original data stream However while the packets are in transmit the network does not know that there is any connection between them It is perfectly possible that packet(n) will actually arrive before packet(n 1) It is also possible that somewhere in the network an error will occur and a packet will not get through at all In that case that packet has to be sent again TCP is responsible for breaking up the message into packets reassembling them at the other end resending anything that gets lost and putting them back in the right order

The IP layer is responsible for routing the individual packets In an internet simply getting a packet to its ultimate destination can be complex job A connection may require the packet to go through several networks eg through ethernets 56K baud phone lines etc Keeping track of the routes to all the destinations and handling incompatibilities among different media must be done at this layer Note that the

interface between TCP and IP is fairly simple TCP simply hands IP a packet with the destination s IP address IP does not know how this packet relates to any packet before it or after it

It is not enough to get a packet to the right destination as there may be multiple connections even between the same pair of machines TCP has to know the connection to which a received packet belongs This task is referred to as demultiplexing In fact there are several levels of demultiplexing going on in TCP/IP The information needed to do this demultiplexing is contained in a series of headers A header is simply a few extra octets tacked onto the beginning of a packet by some protocol in order to keep track of it

Consider a large data stream to be sent from one computer to another TCP finds how large the packet should be for the network to handle breaks the stream into packets of that size and puts a header at the front of each packet The TCP header is shown in Figure 3 1 This header contains at least 20 octets but some of the most important ones are the source and destination port numbers and the sequence number The port numbers are used to keep track of different connections TCP allocates a port number for a particular connection The process sending a packet identifies a source port as the source of the packet The TCP layer at the other end would have assigned a port number of its own for this connection The TCP layer at the source has to know the port number used by the other end as well and will keep this information in the destination port field Each packet will also have a sequence number This is used so that the other end can make sure that it gets the packets in the right order and that it has not missed any packets in between TCP actually numbers the octets of the data stream For example if there are 500 octets of data in each packet then the first packet will be numbered 0 the second 500 the next 1000 1500 and so on

In the TCP header the Check sum field is a number that is computed by adding up all the octets in the packet The result is put in the checksum field The TCP layer at the other end computes the check sum again If they disagree, then the packet is assumed to have errors and is discarded The format of the TCP header is shown in Fig 3 1

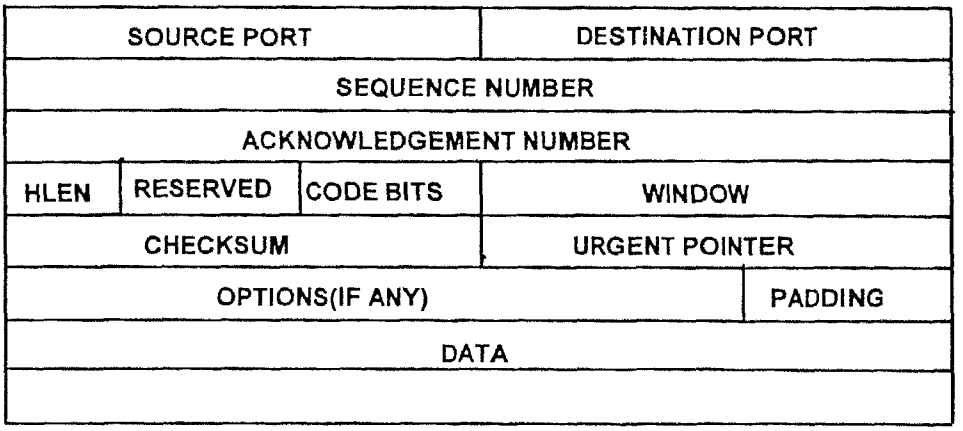


Figure 3 1 TCP Header

The TCP layer appends its header to the stream of bytes and forms the corresponding packets It sends each of these packets to the IP layer The IP layer has to know the internet address of the computer at the other end Note that this is all that IP is concerned about It does not care what is there in the packet or even in the TCP header IP s job is simply to find the route for the packet and get it to the other end In order to allow routers and other intermediate systems to forward the packets it adds its own header to the packet before transmission The IP header is shown in Figure3 2

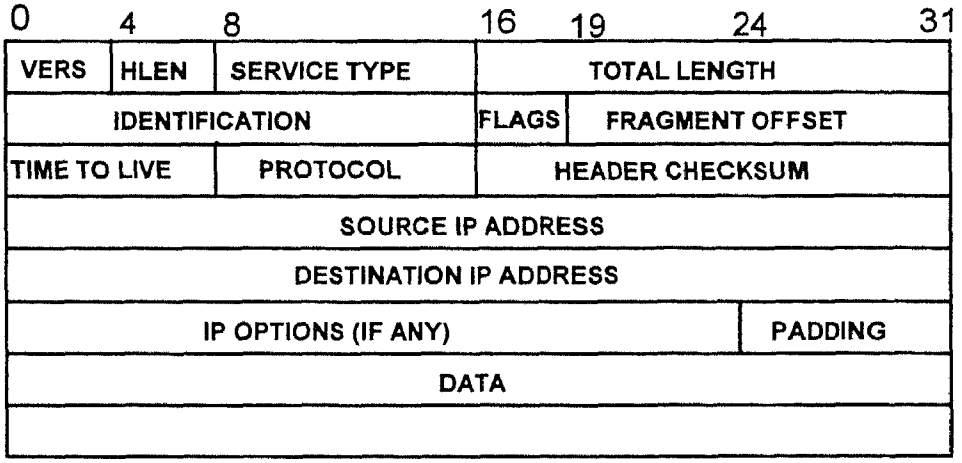


Figure 3 2 IP Header

The main fields in this header are the source and destination internet addresses the protocol number and another checksum The source internet address is the address of the source The destination address is the address of the other machine The protocol number tells the IP on other end to send the packet to TCP (Although most

IP traffic is used by TCP there are other protocols that use IP as well which makes this field necessary) Finally the checksum allows the IP layer at other end to verify that the packet was not damaged in transmit Note that TCP and IP have separate checksums This is because IP does not know anything about TCP As far as IP is concerned everything after its header is a sequence of bits IP will therefore compute its own checksum to make sure that the IP packet did not get damaged in transmit

3 2 Linux Operating System

Linux kernel is the software which provides the minimum utilities necessary for implementing the operating system facilities The main modules of the linux kernel are the *memory management* support *file system terminal handling* support *interprocess communication network communication* support and some basic facilities such as timer and system clock handling descriptor management and process management The *memory management* module deals with the task of allocating the address space to the application programs and does paging and swapping The *file system* manages the files directories pathname translations file locking and I/O buffer management The *terminal handling* support controls the input character editing and the output delays for the standard system I/O operations The *interprocess communication* deals with the management of the sockets The *network communication* support deals with the implementation of the Communication network protocols such as TCP/IP Appletalk Ax 25 etc

We have used the complete linux network source code for the implementation of our router We have used the TCP/IP network software implementation of the Linux Kernel Linux Kernel networking software has been designed for use either as hosts or as routers With IP Forwarding enabled a Linux host can be used as a router The internals of the networking software which we have used for our router implementation is described in the following sub sections

3 2 1 Networking Support

In this subsection we consider the internals of the implementation of TCP/IP software which are of concern to our implementation here The data which is to be sent from source machine to the destination machine is first segmented the TCP and IP headers are added and the data is transmitted in the form of packets (datagrams) These datagrams are sent through various routers and intermediate networks until they reach the destination machine This has been illustrated in the following figure

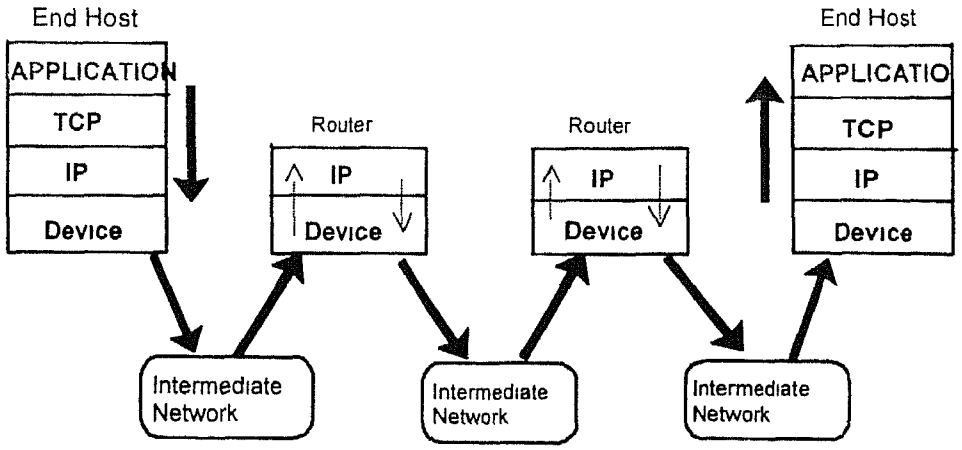


Figure 3 3 Transfer of datagram over an Internet

An intermediate router in the path will take in the incoming packet and will check its routing table for an entry which specifies the interface over which this packet has to be sent It then forwards the packet to this device

The following figure gives the internals of the router implementation

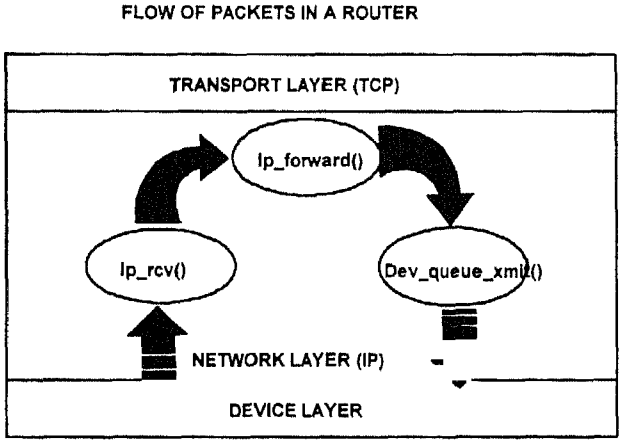


Figure 3 4 Internel of routing module

The router device driver receives the packet from the network interface It processes the packet and when it finds that the packet is for the IP protocol it hands it to the **ip_rcv()** routine The **ip_rcv()** routine processes the packet in one of four ways (a) it is passed as an input to a higher level protocol (b) if it encounters an error it reports back to the source (c) it is dropped because of an error or (d) it is forwarded along the path to the destination The **ip_rcv()** first checks whether the datagram is acceptable or not It checks the version and drops the packet if this does not match (The current version is ipv4 which is 4) It calculates the checksum of the IP header and discards the packet if there is an error It then checks whether the packet length is at least the length of IP header and also processes any options in the IP header The packets which are received by a router have to be sent to the appropriate network interface they will not be sent to the upper layer protocols When **ip_rcv()** detects that the packet is not for this host it performs IP packet forwarding For this the IP Forwarding has to be enabled and the **ip_rcv()** routine then hands the packet to the **ip_forward()** routine

The **ip_forward()** routine first decreases the time to live (ttl) field in the IP header of the packet When this reaches zero it drops the packet and sends an ICMP control message indicating that the packet s lifetime has expired It recomputes the check sum of the IP header Then **ip_forward()** calls **ip_rt_route()** to get the routing table entry for the destination address in the IP header The routing table consists of details such as the interface via which the packet has to be sent and whether there is a default gate way available for this If the routing table entry has a default gateway entry then it sends the packet to that particular gateway otherwise it sends the packet directly to the destination host on the destination net Then **ip_forward()** calls **dev_queue_xmit()** and passes the following three arguments to it the packet the device and the type of service that is requested by the packet The routine **dev_queue_xmit()** is a generic one which is used by all the protocols It takes the responsibility to hand it to the device driver transmit routine Each device has three

send queues which may be used to buffer the packet The way in which the queues are build up and serviced by the device has been explained in Section 3 2 4

3 2 2 IP packet Filters and Firewall support

The Linux Operating system includes a number of facilities for efficient kernel level IP packet filtering and screening The IP packet filtering routines may be used to implement the Firewall utilities A Firewall is a network security device which sits between the local network and the Internet It regulates the access of the local net to the internet and vice versa A Linux router may also be used as a firewall by enabling the Firewall options and by specifying some filter rules IP packet filters accept and forward IP packets by inspecting these filter rules These rules can be specified by using packet characteristics such as IP address port numbers and IP flags and by device characteristics such as the incoming and outgoing interfaces Linux supports masquerading for the forwarding packets so that all packets appear as though they come from the Linux host The IP packet filters may be implemented at three stages These three stages are the Input filter the Forwarding filter and the Output filter At each filter the rules may be specified with each having a policy for the packets The policy may be accept reject or deny Accept will pass the packet through the filter whereas deny and reject will drop the packet Reject will in addition send back an ICMP reply message specifying that the destination is unreachable The decision to let a filter accept reject or deny the packets is usually based on several criteria which will be checked against the contents of the IP packet and some environmental parameters The various criteria used by the filters are the Source and Destination IP addresses Protocol IP options Source and Destination port numbers and the Network Device via which a packet is received or is going to be sent The rules are to be set specifying all the details such as which filter to use the policy the source and destination networks the protocols if any optional via device ports associated with TCP or UDP services etc These filter rules are set in the kernel by using the command **ipfwadm** This is a utility to administer the IP accounting and IP firewall services offered by the Linux kernel The features offered by **ipfwadm** are

- Changing the default policies for all firewall categories
- Automatically adding the necessary extra rules when the named hosts have more than one IP address
- Support for specifying the interface address for the rules
- Support for specifying the interface name for the rules
- Support for masquerading in the forwarding firewall

The **ipfwadm** command fills the three firewall chains **ip_fw_in_chain, ip_fw_out_chain ip_fw_fwd_chain** for each of the input output and forwarding filters Each chain is a linked list of structures of type **struct ip_fw** This structure contains all the details which are input with the **ipfwadm** command The details such as source and destination network address network masks policy protocol ports via device flags etc The following figure gives the details of the IP packet filtering

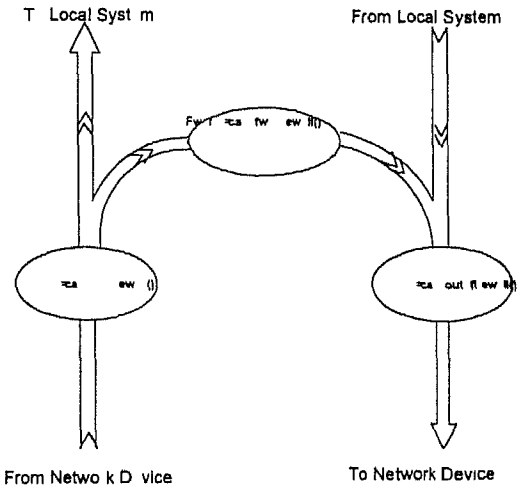


Figure 3 5 IP Packet Filtering

The packets which are received from the network device are passed through the IP input filter The IP input filter is **call_in_firewall()** Similarly the forwarding and output firewalls are **call_fw_firewall()** and **call_out_firewall()** respectively All these functions in turn call **ip_fw_chk()** by passing their respective firewall chains i e **ip_fw_in_chain, ip_fw_fwd_chain, ip_fw_out_chain** Each chain is a list of filter

rules The **ip_fw_chk()** function steps through the rules to check whether the current packet matches with the rule or not The first matching rule (if any) determines the further actions In case of match the rule s policy will be returned This policy which may be accept or reject or deny will then be applied to the packet If none of the filter rules match with the packet it uses the default policy associated with the filter The forwarding firewall supports masquerading The masquerading utility and its internals are discussed in the next subsection

3 2 3 The Masquerading Utility and its Internals

In our router implementation we have used the forwarding IP filters for masquerading Masquerading is done in the forwarding function (*ip_forward()*) and has to be specified in the forwarding filter rules by the **ipfwadm** command In the IP Forwarding Filter the firewall response (policy) will now be *masquerade* instead of *accept reject* or *deny* The packet is passed to the **ip_fw_masquerade()** function whenever the firewall policy is *masquerade* Demasquerading is the reverse process of masquerading and is done by **ip_fw_demasquerade()** The masquerading functionality is shown in Figure 3 6

Through masquerading we can make the packet forwarding such that some or all of the packets being forwarded by the Linux system will be changed so that it would appear as if they are being sent by the local system The source IP address is replaced by the interface address via which the packet is to be sent and the port number is replaced by another locally generated port number The Linux kernel assigns masquerading ports starting from 61000 The range of the masquerading ports is 61000 614096 For each new connection it assigns a new port When the masquerade port reaches 614096 it rolls back to 61000 The application servers in the internet think that the packets are generated by the masquerading host and hence send the packets back to it The masquerading host is also responsible for demasquerading the return traffic i e to send the packets of the return traffic to their respective hosts In order to demasquerade the packets two tables are maintained Each entry in the table contains a seven tuple along with information for linking the entries in the table The

seven tuple is < *protocol source address source port destination address destination port masquerading address masquerading port* > At the time a connection is being initialized an entry in the two tables **ip_masq_s_tab** and **ip_masq_m_tab** is made The seven tuple will then be used to masquerade and demasquerade the packets of this connection The following figure shows both masquerading and demasquerading routines

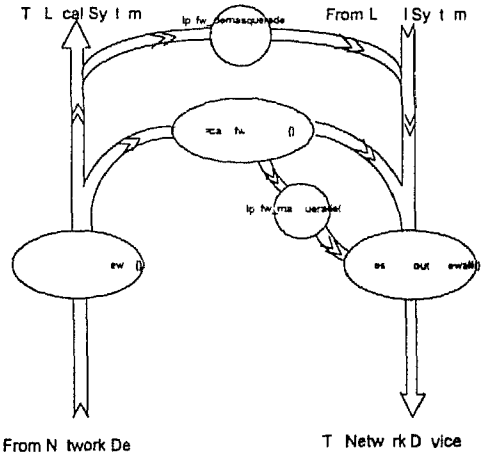


Figure 3 6 Masquerading Functionality

For a client machine on the local network to connect to the internet via the masquerading host it has to first set the forwarding rule with **ipfwadm** on the command line as follows

 # ipfwadm F a masquerade **S** localnet/netmask **D** anywhere/netmask

The above example is a command for setting up the firewall rules with minimum options There are many other options for **ipfwadm** for specified utilities In the above example the option **F** sets the rule for the forwarding filter Option **a** appends this rule to the forwarding firewall chain (**ip_fw_fwd_chain**) in the kernel (There are options to insert and delete the rules) The option **S** is for the network whose packets are to be masqueraded and **D** is for the destination net (most commonly this will be the complete Internet) Both the network arguments must be name resolved This filter rule specifies that the packets from the local net to anywhere should be masqueraded The **ipfwadm** command takes the protocol ports and the device as

other options the masquerading can therefore be done both on specific protocols and on specific ports as desired The device via which the packets are to be masqueraded can also be specified The following example shows an example with more options

ipfwadm F i masquerade **P** tcp **S** localnet/netmask sport1 sport2 sport3 **D** remotenet/netmask dport1 dport2 **V** ifaddress

The above command inserts a rule in the forwarding filtering rules specifying that data corresponding to only the TCP protocol and for dport1 dport2 on the remote network and from the sport1 sport2 sport3 on the local network going via the interface with address *ifaddress* should be masqueraded The number of ports specified in the command is variable (The *ipfwadm* supports up to ten ports The man page of *ipfwadm* gives all the options in more detail)

At the start of a connection the first packet is given to the masquerading module It assigns an unused port (in the range 61000 614096) and creates an entry in the two masquerade tables **ip_masq_s_tab** and **ip_masq_m_tab** From now onwards the packets (both to be masqueraded and demasqueraded) use these two tables An IP packet going from the localnet to the internet checks for its entry in **ip_masq_s_tab** and keeps the masquerading address and the masquerading port in place of the source address and source port in the IP header and sends the packet to the internet Similarly an IP packet coming from the internet to the local network reads its corresponding entry from the **ip_masq_m_tab** and recovers from it the IP address and port of the host to which it has to forward the packet When the connection is closed the entries in the two tables are removed and the masq port is now free to be used for some other connection

3 2 4 Buffering Mechanism in the Device Queues

Each Network Device has three send queues associated with it Each queue has been given a priority and the queue with the highest priority is serviced first In the Linux Operating System queue 0 is given the highest precedence queue 1 is given the next priority and queue 2 is given the lowest priority The routines **dev_tint()** and **do_dev_queue_xmit()** do all the buffer management in the network device queues

The three queues will be built up when the network becomes congested or the device is too busy servicing the receive packets The routine **dev_tint()** is used to service the packets in the send queues of the device whenever the device is free The routine **do_dev_queue_xmit()** is the interface of the device layer to all the protocols in the network layer The functionality of **do_dev_queue_xmit()** is given in the following flow diagram

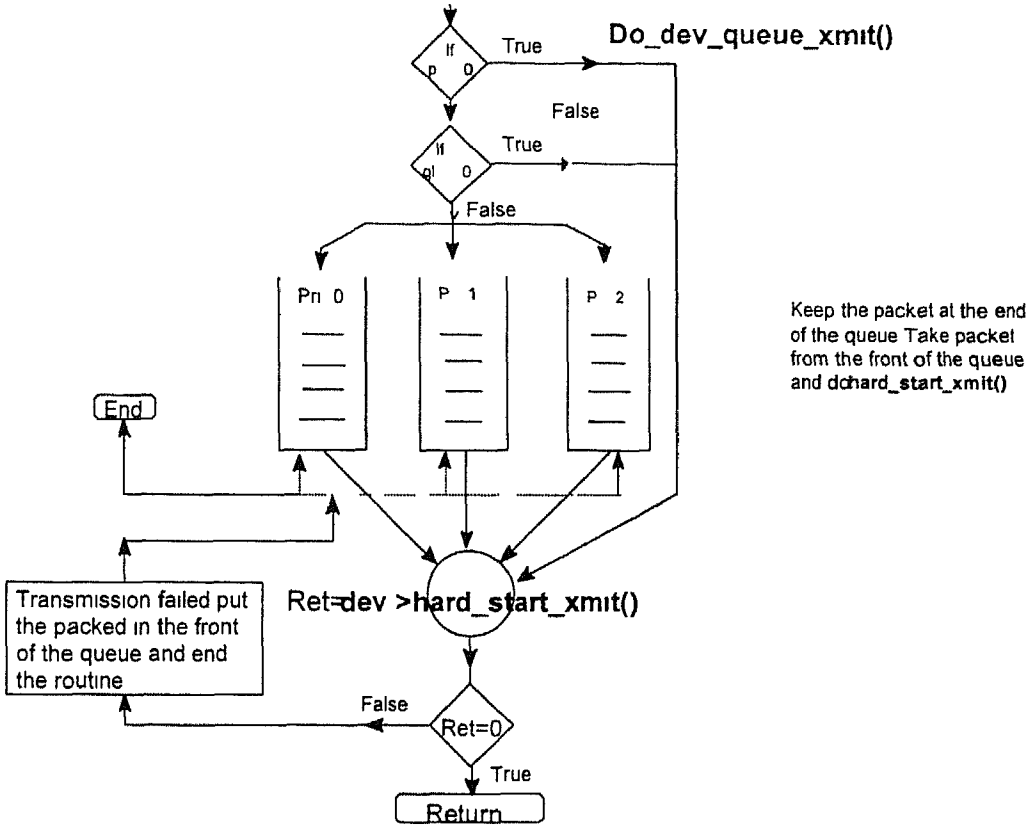


Figure 3 7 Flow of packets in the device queue

The packet along with the priority is passed on to the **do_dev_queue_xmit()** Priority 0 is given the highest priority and is set for packets which have requested for low delay Priority 1 is for packets with high throughput and is given the next precedence Priority 2 has the lowest precedence and is for packets which have not requested either for low delay or for high throughput When the priority value Pri is negative the packet is not put in any of the network device queues and is directly given to **dev >hard_start_xmit()** which is the device transmit function (shown in Figure 3 7) For packets with positive priority if the queue (for that priority) is found

to be empty then the packet is given to **dev >hard_start_xmit()**, otherwise the packet is kept at the end of the queue While servicing the queue the first packet is taken from the head of the queue and is passed to the **dev >hard_start_xmit()** (These queues are FCFS in nature) The function **dev >hard_start_xmit()** returns 0 if the transmission is successful If the transmission fails the packet is kept at the front of the queue of the particular priority and the routine **do_dev_queue_xmit()** ends Only one packet is served when **do_dev_queue_xmit()** function is called If the transmission fails then the packet is once again kept in the front of the queue The transmission fails when the device is busy servicing the receive packets or if it is servicing higher priority packets This implies that when the network is in the Start Of Congestion (SOC) state or in the Network Congestion (NC) state the queues will start building up These queues are serviced by the **dev_tint()** function The flow diagram of the **dev_tint()** function is given in the following diagram

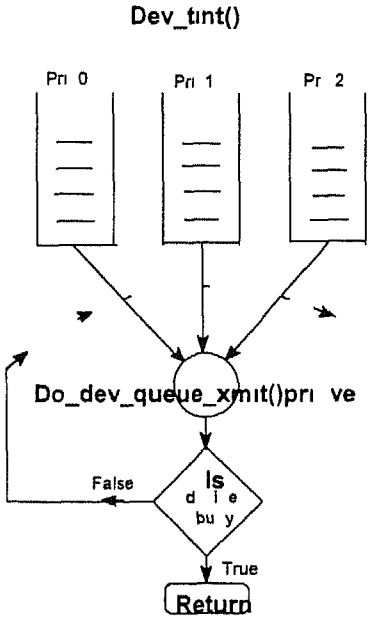


Figure 3 8 Device queue service mechanism

The **dev_tinit()** function will be called whenever the device is free This function services the queues which are built up when the network is in congestion state The first queue with priority 0 has the highest priority and is serviced first Each packet is taken from the front of the queue and is sent to **do_dev_queue_xmit()** by negating

the priority When the priority is negative the packet is directly given to **hard_start_xmit()** of the device as shown in Figure 3 7 After each packet is serviced it checks whether the device is busy If the device is found to be busy if returns form the **dev_tint()** function Otherwise it continues to pass packets to the **do_dev_queue_xmit()** routine After the first queue is emptied it repeats the same service to the second queue and then to the third queue The routine ends after it empties the three queues it will also return from this function if it finds the device to be busy in any intermediate step

3 3 Design and Implementation of the Asymmetric Router

The asymmetric router masquerades the packets coming from the client machines on the user network and sends them via the modem connection It receives the packets via the satellite interface demasquerades and sends them to the respective client machines To achieve this functionality the asymmetric router differs from the ordinary masquerading host (or router) in that *it masquerades the packets via the modem connection but represents to the external world as though they are masqueraded via the satellite connection* Hence the external network will route back the return packets to the satellite interface We have used the information stored in the firewall chain to obtain this functionality

An algorithm is designed and implemented to drop the acknowledgments when there is a congestion in the network A device send queue is added to the existing three device queues and this queue is used to place and process the acknowledgments An acknowledgment from the local network to the internet application server is forwarded at the asymmetric router Then the acknowledgment queue (fourth queue) is checked for any pending acknowledgments for that particular connection If there are any they are droped and the current acknowledgement is sent for transmission If the transmission fails the current acknowledgement is kept back in the fourth queue

3 3 1 Masquerading implementation in asymmetric router

Masquerading of a particular network is done by specifying the forwarding filter rules These filter rules (which are specified by the **ipfwadm** command) are filled in

the **ip_fw_fwd_chain** This is a chain of structures of type **struct ip_fw** Each firewall rule information is stored in a **ip_fw** structure which contains the details like the policy of the rule source network address and netmask destination network address and netmask ports the via device etc This information is used by the Linux Kernel to decide what policy is to be applied on the packet We decided to make use of the information in this structure to do the decision as what is to be the return path When the filter rule is set by the **ipfwadm** command all the information given as arguments is saved in a **ip_fw** structure and this structure is either appended or inserted depending on the option given to the **ipfwadm** command

As shown in Figure 3 6 the masquerading is done at the forwarding firewall The routine **ip_forward()** calls the forwarding firewall **call_fw_firewall()** The routine **call_fw_firewall()** inturn calls **ip_fw_chk()** by passing the forward firewall chain **ip_fwd_chain** The routine **ip_fw_chk()** runs through the complete chain and in case it finds a match it returns the policy of that particular rule This will be applied on the packet In the masquerding case the policy returned will be masquerade and the routine **ip_fw_masquerade()** is called The **ip_fw_masquerade()** routine makes an entry in the masquerading tables at the time of connection establishment In the masquerading entry the *masquerading address* field is filled with the IP address of the device Here we used a routine **ip_fw_chk1()** to fill the desired address in the *masquerading address* field of the masquerading table entry The **ip_fw_chk1()** routine (which does the functionality of **ip_fw_chk()** routine) runs through all the rules and incase any rule matches with the packet the corresponding **ip_fw** structure is returned As mentioned earlier this structure contains all the details corresponding to a particular rule We use the **fw_via** member of the structure This contains the address via which the packet has to go If we keep the High Speed Down Link (HSDL) interface address in this structure (entered with **ipfwadm** command) we can use it to fill the masquerading tables The address of the HSDL interface is extracted from the structure and is filled in the masquerading tables at the starting of the connection So from now onwards the packets which are to be masqueraded take this address and keep it in the source address of the packet Then the packet is sent via the

modem connection We have successfully made the source address of the packet as the HSDL interface The rest of the world routes the packet back to the satellite interface i e the hybrid gateway server when it has to send the data looks from where the the packet came It sees the HSDL interface address and hence routes it to the high bandwidth sub network for transmission

These packets are received by the HSDL interface and here they are demasqueraded The asymmetric router checks the masquerading tables and extracts the corresponding entry It gets the original source address and source port and keeps it in the IP header and sends it to the user network interface

3 3 2 Acknowledgment dropping

Our primary aim is to drop the pending acknowledgments when the network becomes congested For this we developed the following algorithm

- Initialize one more device send queue along with the existing three queues
- Assign the least priority to this fourth queue
- The ack bit in the TCP header is to be checked to confirm if the packet is an acknowledgement or not If the packect is an acknowledgment and if it is not piggy backed the syn and the fin bit is not set do the following else treat the packet as an ordinary packet
- When an acknowledgement comes the fourth queue is checked whether there are any pending acknowledgements for that particular connection
- If there are any and if the acknowledgement number of the current packet is greater than the acknowledgment number of the packets in the fourth queue then drop the packets in the fourth queue and transmit the current acknowledgement packet Otherwise drop the current packet and transmit the acknowledgement with the highest acknowledgment and set a flag indicating that it is a *retransmission*
- If the transmission of the acknowledgement packet fails then keep the packet back in the fourth queue If the *retransmission* bit is set keep the packet in the first queue

In the above we have kept the least priority for the fourth queue intentionally because the congestion is primarily due to a large generation of the acknowledgments If we give high priority to this queue the device driver will spend all its time servicing the acks and other services will be effected This is the reason for the selection of the least priority for the acknowledgment queue TCP supports piggybacking i e that is some of the acknowledgement packets carry data in them We do not want to drop these piggy backed packets At the same time we do not drop the acknowledgements which are used in the connection establishment and connection close as we do not want our acknowledgment dropping effect the establishment and closing of the connection

We drop the pending acknowledgements if the current acknowledgement number is more than that of the pending acknowledgements If acknowledgement number is less than that of the packets that are queued then this is a retransmission So we drop the current packet and transmit the packet with high acknowledgment number If the transmission fails the packet is kept in the first queue (at the front) in case it is a retransmission and in the fourth queue (at the front) if it is not a retransmission We keep the retransmission packet in the first queue because we know that there is a retransmission and we want to process the packet quickly and highest priority is given

The implementation of our algorithm is done in the same lines in which the kernel manages and processes the device queues The internals of the device send queue management is shown in Figure 3 7 and Figure 3 8 The two routines **do_dev_queue_xmit()** and **dev_tint()** are modified and enhanced to implement the acknowledgment dropping At the system bootup time a fourth device send queue is initialized The routine **ip_forward()** calls the function **ack_check()** before passing a packet to the device routines The function **ack_check()** used to check whether the current packet is an acknowledgment to be dropped It checks for the ack bit in TCP header to find whether the packet is an acknowledgement or not The routine **ack_chk()** returns true if the acknowledgement is not piggy backed and neither of the syn or fin bits are set If true is returned by **ack_chk()**, then the priority is set to 3 and

the packet along with device and priority is passed to the modified **do_dev_queue_xmit()** function

The modified **do_dev_queue_xmit()** does the normal processing for the packets with priority from 0 to 2 If the priority is 3 it drops the pending acknowledgments (if any) for that particular connection (The dropping of the acknowledgments per connection is done by keeping a separate record of the acknowledgements on connection basis The packets of each connection are hashed in a separate linked list So we can process the corresponding linked list for the a particular connection When a packet is dropped the corresponding entry is removed from the hash table) It drops the acknowledgments only if the acknowledgment number of the current packet is greater than the acknowledgement of the packet in the queue else (this is a case for the retransmission) the current packet is dropped and the packet in the queue is transmitted (by calling the **dev >hard_start_xmit()**) If the transmission fails then the packet is kept in the first queue in case of retransmission else it will be kept in the fourth queue

The modified **dev_tint()** does the same functionality of the original **dev_tint()** expect that it now services four queues instead of three queues and when if services the fourth queue it unhashes the entry in the hash table which we hashed for the processing of the acknowledgments

Chapter 4

Performance Testing of the Asymmetric Hybrid Router

We have set up a test bed to create a Hybrid Network environment where the performance of our asymmetric router software can be tested In this test bed we have configured different machines to reflect the functionality of the Internet Service Provider and of the hybrid gateway server In our set up we have used the IIT/K s internal telephone network as the low bandwidth upstream channel The return path is a 10 Mbps ethernet cable We have also introduced a delay in the return path This delay can be set from the command line along with the command for configuring the network device The Point to Point Protocol (PPP) is used to connect the user network to the external network using dial up telephone lines We used the Squid cacheing proxy to do TCP connection splitting (TCP spoofing) at the hybrid gateway server We have tried to mimic as much of the functionality of the Internet Service Provider (ISP) as will be relevant for our testing In our setup the ISP has the capacity to assign an IP number to the masquerading host whenever the PPP link is established We have implemented Dial On Demand (at the asymmetric router) this takes care of the function of dialing out to the ISP whenever the link is down (i e when a client machine on the user network wants to access the external world) The test bed setup has been shown in Figure 4 1 The Kernel on each of these machines needs to be compiled after configuring them for their respective functionality The rest of the chapter covers the configuration of various machines of the test bed The initial sections of this chapter cover the implemetation issues such as compiling the

Kernel setting up and using the PPP link to the ISP handling multiple ethernet cards etc The later sections cover the configuration and description of the various machines on the test bed

4 1 Configuring and Compiling the Linux Kernel

The Linux Kernel has to be configured first before it can be compiled This configuration basically involves the selection of proper peripherals filesystems network devices etc and enabling various utilities such as neworking support communication protocols and protocols (like PPP and SLIP) Generally only the basic ultilities required are selected so that the kernel remains small in size The Linux Kernel provides modular support i e devices filesystems etc can be loaded into the kernel as modules

The Kernel configuration and its compilation has to be done with super user privilege The Kernel source code is kept in the /usr/src directory There will be a directory by name linux in /usr/src if kernel source code had been installed when Linux Operating system was first set up If it has not been installed or if a new version is to be installed (i e a new kernel with changes as for our requirements) then the source code needs to be unpacked into /usr/src from the original distribution This can be done with the command tar xvzf linux x y z tar gz

The directories /usr/include/asm /usr/include/linux and /usr/include/scsi should be properly symbolic linked to the kernel sources The following may be done to ensure this

```
# cd /usr/include
# rm rf asm linux scsi
# ln s /usr/src/linux/include/linux linux
# ln s /usr/src/linux/include/asm i386 asm
# ln s /usr/src/linux/include/scsi scsi
```

The stale o files and dependencies lying around the kernel source code should be removed if we want to start with a fresh kernel The following may be done for this

```
# cd /usr/src/linux
```

make mrproper

The kernel may be configured once the code has been cleaned This can be done by executing the make config command Including unnecessary drivers will make the kernel bigger and may also lead to other problems (e g probing for a nonexistent controller card may confuse other controllers) Compiling the kernel with Processor Type set higher than 386 will result in a kernel that will not work on a 386 machine The kernel will detect this at boot and will give up The make config program starts a configure script which asks several questions these need to be answered usually with either y or n The correct options have to be selected for the general setup filesystems character devices networking options network device support and for some other specific options like CD ROM drivers and SCSI support (in case the system supports them as well) The configure script finishes after all the options have been properly selected

To set up all the dependencies correctly a make dep has to be done We can also do a make clean to remove all the object files and other files that an old version may leave behind After these have been done the source code may be compiled with make zImage This compilation will generate a new compressed kernel called zImage which will be found in the appropriate boot directory

The new kernel will work the way it has been configured The Linux Loader (LILO) may be used to install it this would usually be installed when the Linux Operating System is first installed The configuration file /etc/lilo conf contains all the information regarding the kernel images and the current setup The config file looks like this

```
image = /vmlinuz
label = Linux
root = /dev/hda1
```

The image = is set to the currently intalled kernel which in this case is /vmlinuz The label is used by LILO to determine the kernel or operating system to boot from and root is the block device (eg hard disk partition of the OS in a PC) of that particular operating sytem One should make a backup copy of the old kernel and

copy appropriately the new kernel zImage which has just been made (This can be done as cp /usr/src/linux/arch/i386/boot/zImage /vmlinuz in our PC based system if we use /vmlinuz as the image) LILO is then run on running it will display all the images that it has added The system can now be rebooted with the new kernel

4 2 The Point to Point Protocol (PPP)

The Point to Point Protocol is a mechanism for creating and running IP and other network protocols over a serial link A Linux machine (e g a PC) can use PPP to connect to a PPP server and use this to access the resources of the network to which the server is connected this will work as though the Linux machine is directly connected to the network to which the the PPP server is connected A Linux machine can also be configured as a PPP server so that other computers can dial into it and access the resources of the local network There is techinically no difference between the machine that dials in and the machine that is dialed into However for clarity s sake it is useful to think in terms of servers and clients A machine dialling into a site to establish a PPP connection is referred to as a client The machine to which the connection is established is referred to as the server

4 2 1 Serial Line

A machine which wants to communicate using modems and telephone lines must have a serial port (with the appropriate software driver) to which the modem will be attached Normally the serial port used for this in a PC will be the COM1 port For this port the corresponding serial device is /dev/ttyS0 In case this device is not present it should be created as

```
# mknod  m 755 /dev/ttyS0 c 4 64
```

or by doing

```
# cd /dev
# MAKEDEV ttyS0
```

In a PC this device must also be assigned the serial device interrupt number and the corresponding address A PC normally has ttyS0 and ttyS2 at IRQ 4 and ttyS1

and ttyS3 at IRQ 3 In this case the required assignement may done with the SETSERIAL command

/sbin/setserial /dev/ttyS0 auto_irq skip_test autoconfig

This command will initialize the ttyS0 device The use of auto_irq tells the kernel to try to automatically determine the Interrupt Request Number (IRQ) during the autoconfiguration The parameter skip_test tells the kernel to skip the UART (universal asynchronous receive and transmit) test during autoconfiguration The parameter autoconfig tells the kernel to attempt to automatically configure the serial port With this parameter the kernel tries to automatically determine the UART type and its parameters This needs to be done every time the system is booted this is probably best done through a start up file such as /etc/rc d/rc serial This file is usually also invoked from /etc/rc d/rc S This rc S file is invoked by init at boot time by specifying it in the /etc/inittab file If hardware flow control is to be used then the following command should also be added in /etc/rc d/rc local file

#stty crtscts < /dev/ttyS0

The file rc local is also invoked from the /etc/rc d/rc S file at boot time

The PPP server needs some additional configuration The Linux Operating System has a getty process listening on each of the terminals to be used for logging in This process is forked by init The process gettty displays the login banner It accepts the user name displays the password prompt accepts the password and then checks for an entry in the /etc/passwd file for the matching login password pair If a match is found then home directory is set to the path specified in the corresponding line entry in passwd file It then over lays upon itself the shell that has been specified When the user shell dies away (i e when the user logs out) a new getty is spawned by the init process Similarly a process should be listening on the PPP server machine for a PPP client machine to dial in This process should be listenimg on the serial port whose device file is /dev/ttyS0 An example of such a process is the mgetty process for which the following entry should be made in the /etc/inittab file

s1 12345 respawn /usr/local/sbin/mgetty ttyS0 38400 vt100

For both the PPP Client and the PPP Server the serial line also needs to be initialized On the PPP Server there should be a mgetty line listining for a PPP client to dial in Both the client and the server machines need to be configured with the PPP support and should then be compiled with this Kernel configuration and compilation had been explained in detail earlier in Section 4 1

4 2 2 Configuration of PPP Client

The PPP Client dials out to connect to a PPP Server This is done by invoking pppd with the appropriate options The command pppd accepts various options which may be used to configure the PPP link depending on the environment in which it is to be operated The pppd command can be given various arguments in the command line itself These options can also be given in the /etc/ppp/options file in which various parameters can be specified The parameters in the options file are read before the command line arguments are parsed and would therefore override the latter

The pppd uses a chat program for a conversational exchange between the computer and the modem The primary purpose of the chat program is to establish a connection between the local pppd and the remote pppd The chat script defines how the communication should take place A script consists of one or more expect send pairs of strings with an optional subexpect subsend pair separated by a dash as in the following example

ogin BREAK ogin *plogin* ssword *ppassword*

If the above line is given as an argument to the chat command then chat will give the user id *plogin* to the login prompt it receives and will wait for the password prompt to give it *ppassword* as the password This chat script (present in /etc/ppp/ppp on dialer file) is called by the ppp on file to make the connection This ppp on file may be used to automate the ppp connection (This file will usually come with the installation) Information like the account name password and the telephone number to connect to are to be specified in this file This file contains the following line which starts the PPP session

exec /usr/sbin/pppd connect /etc/ppp/ppp on dialer /dev/ttyS0 38400 debug lock modem crtscts asyncmap 20A0000 escape FF kdebug 0

Various options can be added to this line for specific tasks These options can also be specified in the /etc/ppp/options file The command pppd take the line (via which it has to communicate) /dev/ttyS0 as an argument The /etc/ppp/ppp on dialer is the chat script and comes with the installation This may be edited for enhanced use The ppp off' file can be used to close the PPP session This file will kill the running pppd process it will also remove any lock file created by pppd so that this device can be used by other processes or can be used again by a new PPP session This is necessary as otherwise pppd cannot be run again i e the system will give the error message resource or device busy if this has not been done

4 2 3 Configuration of the PPP Server

The PPP Server is to be configured to permit another machine (PPP client) to dial in and start a PPP session On the PPP Server the serial line should be initialized and it should be listening to accept incoming connections In Sec 4 2 1 we have given details about the Serial Line and its setup The mgetty daemon should be running on the PPP Server for accepting connections from the client machines

The PPP Server has one user account for the PPP clients to log in The PPP Client is provided with this account id and its password Whenever a PPP Client logs in by dialing in the pppd (PPP deamon) is automatically started This is achieved by making the *ppplogin* script execute on login The entry in the /etc/passwd file looks like this

plogin encrypted password 513 100 PPP ACCOUNT /temp /etc/ppplogin

As shown in the above line we use /etc/ppplogin as the shell script The ppplogin script looks like this

```
# !/bin/sh
# ppplogin   script to fire up pppd on login
mesg n
stty  echo
exec pppd  detach silent modem crtscts
```

This file automatically fires up pppd when any user properly logs into this account this acts as the corresponding login shell The mesg n command disables other users from writing to ttyS0 using write or any similar commands The stty echo turns off character echoing This is necessary because otherwise everything the peer sends will be echoed back to it In the last line the pppd is invoked with some options The detach option tells the pppd not to detach from the controlling ttyS0 This is very important If we do not specify this option then pppd will go to the background making the kernel shell script to exit This will in turn cause the serial line to hang up and the connection will be dropped The silent option make the pppd wait until it receives a valid LCP (Link Control Protocol) packet from the client before it starts sending LCP packets This prevents the transmit timeouts from occurring when the calling system is slow in firing up its PPP Client The modem option makes the pppd watch the line to see if the peer has dropped the connection and the crtscts turns on the hardware flow control We can also create the /etc/ppp/options file to give other options to pppd

4 2 4 Configuration for Dial on Demand

The Linux host can be configured as a Dial on Demand PPP router i e whenever a computer (on a local network connected to the Linux router) wants to access the internet the Linux router dials out automatically if the link is currently down The kerneld utility can be used to achieve this Kerneld will allow the kernel modules (such as device drivers network drivers filesystems) to be loaded automatically when they are needed rather than having to do this manually with modprobe or insmod

We have to run manually the 'ppp on script to establish the PPP link However if we want to set up our PPP client as a router then we need to have the PPP link automatically established whenever a client machine on the local network needs service For this the kernel has to be configured (and compiled) with the kerneld support to load modules automatically when they are needed and swap them back to the secondary storage when they are no longer needed The following lines should be added to the file /etc/rc d/rc modules file to start the kerneld daemon at boot time

/etc/rc d/rc modules

```
if [ ! -f /lib/modules/`uname -r`/modules.dep ] ; then
    echo "Updating module dependencies for Linux `uname -r`"
    /sbin/depmod -a
fi
[ -x /sbin/kerneld ] && /sbin/kerneld
```

The first part checks for the module dependencies and creates them if they do not yet exist

The kernel now checks in its routing table entry for the route to the packet If the route to a packet does not exist then it calls request route with the address of the desrtination as the parameter The request route file can be used to call pppd and a chat file to dial the other machine The options specified to pppd in the /usr/sbin/ppp on are specified here This file then executes and creates a PPP link The /sbin/request route file should look like

```
#! /bin/sh
LOCK=/tmp/.request route
PATH=/usr/sbin:$PATH       # for ppp-2.2*
export PATH
# Note: you are _not_ forced to use ppp!
# You can do whatever you want in order to satisfy the kernel route request
# It might be a good idea to set up the route as the default route, in case
# you are using e.g. slip or plip or any other net driver
#
# This script will be called from kerneld with the requested route as $1
# Create a chat script for your nameserver (as defined in /etc/resolv.conf)
#
chatfile=/etc/ppp/chat.$1
if [ -f $chatfile ]
then
        #
```

```
        # Tune your favourite parameters to pppd including the idle disconnect
option
        # Kerneld will be automatically triggered to load slhc o and ppp o
        #
        pppd connect  chat  f $chatfile  /dev/modem 38400 \
           idle disconnect 600 modem defaultroute noipdefault \
           &  # let pppd detach itself whenever it wants to

        #
        # Timer to be killed by ip up  tunable! Check kerneld delay as well
        #
        sleep 60 &
        sleepid=$!
        echo $sleepid > $LOCK
        wait $sleepid
        rm  f $LOCK
        exit 0
else
        exit 1
fi
```

The kerneld daemon can be passed certain parameters in the command line The delay can also be increased if the set up of the PPP link takes more time In our local test set up implementation the chat 144 16 160 231 also needs to be created to dial our ISP server This file on our hybrid router machine looks like

```
# !/bin/sh
# /etc/ppp/chat 144 16 160 231
# This is part 2 of the ppp on script It will perform the connection
# protocol for the desired connection
#
        TIMEOUT             3                       \
```

```
ABORT           \nBUSY\r                    \
ABORT           \nNO ANSWER\r                \
ABORT           \nRINGING\r\n\r\nRINGING\r   \
        \rAT                        \
'OK +++\c OK    ATH0                \
TIMEOUT         30                  \
OK          ATDP7230                \
CONNECT                             \
ogin  ogin  plogin                  \
assword     ppassword
```

The following lines should be added to the /etc/ppp/ip up script

```
LOCK=/var/run/request route pid
[ f $LOCK ] && kill 'cat &LOCK'
```

This removes any lock file created by request route

4 3 Configurtation of multiples ethernet cards in Linux

By default the Linux kernel probes only for a single ethernet card and once one is found the probe ceases For probing two or more cards we have to specify this to the Linux kernel so that it can probe them at boot time We can pass these parameters to the Linux kernel at boot time as it can recognize certain parameters passed to it at that time Most often these parameters specify various aspects of the configuration that can not be determined at boot time For network cards the following parameters are passed

ether=<IRQ> <IO ADDR> <PARAM1> <PARAM2> <NAME>

The parameter IRQ is the interrupt request number IO ADDR specifies a single base address to probe PARAM1 and PARAM2 specify the memory of the cards that use shared memory and NAME is the predetermined name of the device like eth0 eth1 etc

LILO (Linux loader) can be used to pass these parameters The LILO boot manager provides two ways to pass these boot time parameters to the kernel The

most common way to do this is to type them immediately after specifying the name of the boot image The following examples enables all the three of the avialable probe slots on a machine

Lilo linux ether=0 0 eth1 ether=0 0 eth2 ether=0 0 eth3

It will be complicated to type this in at each boot The kernel parameters can be made permanent by adding an append line to the LILO configuration file /etc/lilo conf and running lilo to install the updated configuration

append = ether=0 0 eth1 ether=0 0 eth2 ether=0 0 eth3

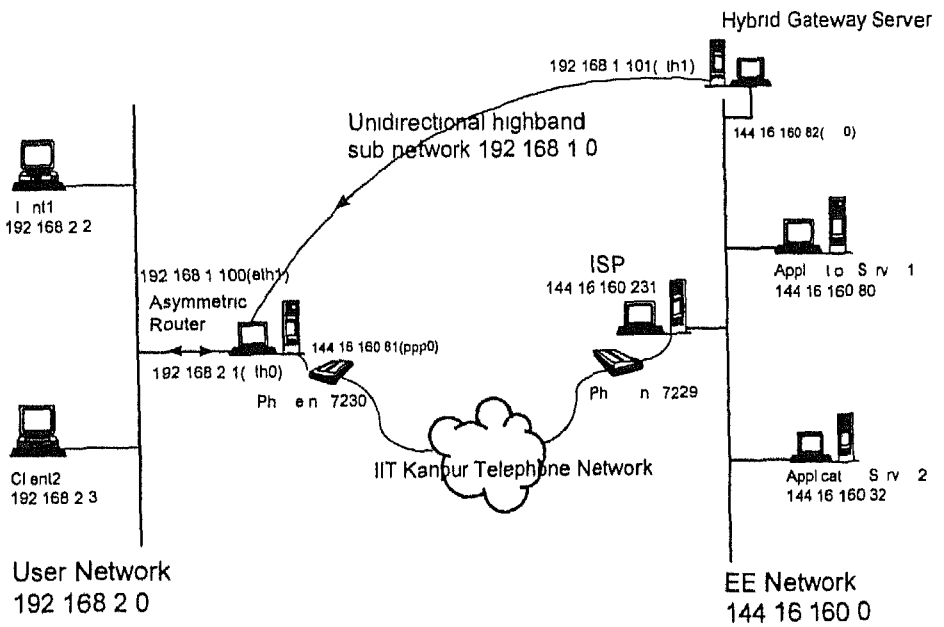


Figure 4 1 The Test Bed Setup

4 4 Configuration of the Masquerading Server/Asymmetric Router

The Asymmetric Router consists of two ethernet interfaces (one for the user network and another for the satellite sub network) and a serial line interface (COM1) for the PPP link We used 8 bit Western Digital ethernet cards To probe the two ethernet cards at boot time(as explained in Section 4 3) we add the following line in the /etc/lilo conf file at the asymmetric router

append = ether=2 0x300 0xd0000 eth0 ether=5 0x280 0xd8000 eth1

We have given interrupt numbers 2 and 5 for the eth0 and eth1 devices respectively

The main function of the asymmetric router is that it does masquerading and routes the packets from the user network interface to the external network via the PPP link This implies that the Linux Kernel of this machine has to be configured with ip forwarding (for router functionality) firewall utilities and masquerading and PPP support and then compiled This machine has to be configured as a PPP client with the Dial On Demand facility (Configuration of PPP client and Dial On Demand have been described in Section 4 2) The options given to the pppd are *noipdefault defaultroute* and *idle disconnect 60* The option *noipdefault* will force the pppd not to take the default IP address of the device The IP address which is given by the PPP-server will be assigned to the PPP interface This is used for the dynamic IP assignment provided by the ISP The option *defaultroute* will add a default route to the system routing tables using the peer (ISP) as the gateway This entry is removed when the PPP connection is brought down This option will make all the up link traffic take the default route via the PPP interface The option *idle disconnect 60* specifies that pppd should disconnect if the link is idle for 60 seconds (The link is termed to be idle when no data packets are being sent or received) The telephone number of the ISP in our test setup is 7229 the assymteric router dials out from the telephone number 7230

For the asymmetric router to do masquerading the masquerading rules need to be set The masquerading modules are to be loaded at boot time The following lines are to be added to the /etc/rc d/rc local file

/sbin/depmod a

/sbin/modprobe ip_masq_ftp

/sbin/modprobe ip_masq_raudio

/sbin/modprobe ip_masq_irc

A few masquerading rules are given below These can be filled in a start up file or can be manually entered when needed

ipfwadm F p masquerade

ipfwadm F a masquerade P tcp S 192 168 2 0/255 255 255 0 D 0 0 0 0/0 8000 V 192 168 1 100

ipfwadm F a masquerade P tcp S 192 168 2 0/255 255 255 0 D 0 0 0 0/0 23 V 144 16 160 81

The first rule sets the default policy to masquerade The second rule specifies that all the connections from the network 192 168 2 0 to any destination network for destination port 8000 (the squid proxy port on the hybrid gateway server) should be masqueraded and that the return path for these should be the High Speed Down Link (HDSL) interface (192 168 1 100 as shown in Figure 4 1) The third rule specifies that all the connections from the network 192 168 2 0 to any destination network for destination port 23 (i e telnet port) should be masqueraded and the return path should be the ppp0 interface (144 16 160 81)

The routing table entry before the PPP link comes up will be

Destination	Gateway	Genmask	MSS	Iface
192 168 2 0	*	255 255 255 0	1500	eth0
loopback	*	255 0 0 0	3584	lo

After the link comes up the routing table will become

Destination	Gateway	Genmask	MSS	Iface
192 168 2 0	*	255 255 255 0	1500	eth0
loopback	*	255 0 0 0	3584	lo
144 16 160 231	*	255 255 255 0	552	ppp0
default	144 16 160 231	*	552	ppp0

4 5 Configuration of the Internet Service Provider

The Internet Service Provider (ISP) should be configured as a PPP Server Section 4 2 gives the configuration details for such a server The serial line should

also be properly initialized This machine is configured to assign an IP address when ever a client machine logs in For dynamic IP assignment the /etc/ppp/options ttyS0 file should have the entry 144 16 160 231 144 16 160 81 for the local IP remote IP pair When the client machine dials in with the noipdefault options then the remote IP will be assigned to the client machine this is 144 16 160 81 in this case

Packets which are destained for the client machine have their destination address as 144 16 160 81 When these come from the externel network to the 144 16 160 0 network they have no route to reach the asymmetric router To overcome this problem we use the proxyarp option at the ISP This option is specified in the /etc/ppp/options file When PPP link comes up the ISP adds an entry to the ARP (Address Resolution Table) with the IP address of the peer and the ethernet address of the system

The routing table before the PPP link come up will be

Destination	Gateway	Genmask	MSS	Iface
144 16 160 0	*	255 255 255 0	1500	eth0
loopback	*	255 0 0 0	3584	lo
default	144 16 160 1	*	1500	eth0

After the PPP link comes up the device ppp0 is loaded and the appropriate routing table entry is added The routing table entry now looks like

Destination	Gateway	Genmask	MSS	Iface
144 16 160 81	*	255 255 255 0	552	ppp0
144 16 160 0	*	255 255 255 0	1500	eth0
loopback	*	255 0 0 0	3584	lo
default	144 16 160 1	*	1500	eth0

4 6 Configuration of the Hybrid Gateway Server

We configured the Squid proxy caching software so that it can do the required TCP connection splitting on the Hybrid Gateway Server Squid is a high performance cache server for the web clients supporting FTP gopher and HTTP data services It does this by accepting requests for the data that is to be downloaded and then handles these requests itself For example to download a web page the user asks Squid to get this page Squid then connects to the remote server and requests the page It then transparently streams the data through itself to the client machine Unlike traditional caching software Squid handles all requests in a single non blocking I/O driven process Squid consists of a main server program *squid* a Domain Name Server lookup program *dnsserver* a program for retrieving FTP data *ftpget* and some management client tools When *squid* starts up it spawns a configurable number of *dnsserver* processes each of which can perform a single blocking Domain Name System (DNS) lookup

We have compiled the Squid on the Hybrid Gateway Server and installed it in the /usr/local/squid directory of that machine Squid creates a few directories squid bin etc cache logs src in the in /usr/local/squid/ directory The /bin directory contains the exectables /src contains the source files /etc contains the squid conf file which is the configuration file and /logs contains various log files The squid conf should be properly edited as per the requirements of the server In this file the http port on which squid operates has been specified to be 8000 The following command should be given to run Squid at the Hybrid gateway Server

/usr/local/squid/RunCache &

This is entered in the /etc/rc d/rc local file so that Squid can be started when the system boots up The Hybrid Gateway Server can now handle the connections on behalf of the client machines on the user net It should have the proper routing table entry to route the return packets to the high speed sub network The routing table at hybrid gateway server should look as shown below

Destionation	Gateway	Net Mask	MSS	Iface
192 168 1 0	*	255 255 255 0	1500	eth1
144 16 160 0	*	255 255 255 0	1500	eth0
loopback	*	255 0 0 0	3584	lo
default	144 16 160 1	*	1500	eth0

As shown above all the packets destined to the 192 168 1 0 IP network will be sent via interface eth1 Interface eth1 is connected to the high speed sub network(as shown in Figure 4 1) The masquerading router keeps the (HSDL) interface address (192 168 1 100) as the source of all the packets Hence for the return packets these become the destination address According to the routing table these will then be routed to the satellite sub network

To simulate an exact test environment for the hybrid network we have introduced a selectable delay in the high speed sub network We modified the **ifconfig** command to introduce delay in the device The modified interface configuration command **ifconfig1** will write the delay in the *device structure* This information will be used to introduce the delay for the device eth1 For example the following command will introduce a delay of 300 milli seconds for the device eth1

/sbin/ifconfig1 delay 300 dev eth1

4 7 Configuration of the Client Machines

The client machines on the user network need very little configuration This is in keeping with our philosophy that the end machines should operate transparently through the Hybrid Network We need to set the default route on these machines to the asymmetric router The machines also have to set their proxy servers as the Hybrid Gateway Server with the following commands

setenv http_proxy http //144 16 160 82 8000/

#setenv ftp_proxy http //144 16 160 82 8000/

The routing table entry at these machines will be the following in our test setup

Destination	Gateway	Net Mask	MSS	Iface
192 168 2 0	*	255 255 255 0	1500	eth0
loopback	*	255 0 0 0	3584	lo
default	192 168 2 1	255 255 255 0	1500	eth0

4 8 Performance of the Test Bed setup

The performance of the Test Bed was measured based on the data avaliable in the *access log* at the hybrid gateway server The *access log* file is the log file generated by the squid process and logs the information such as how many bytes of data transfered to a client machine the total time taken for the transfer etc This time includes the connection establishment time and the data transfer time Using this information we calculated the transfer rate by downloading large files

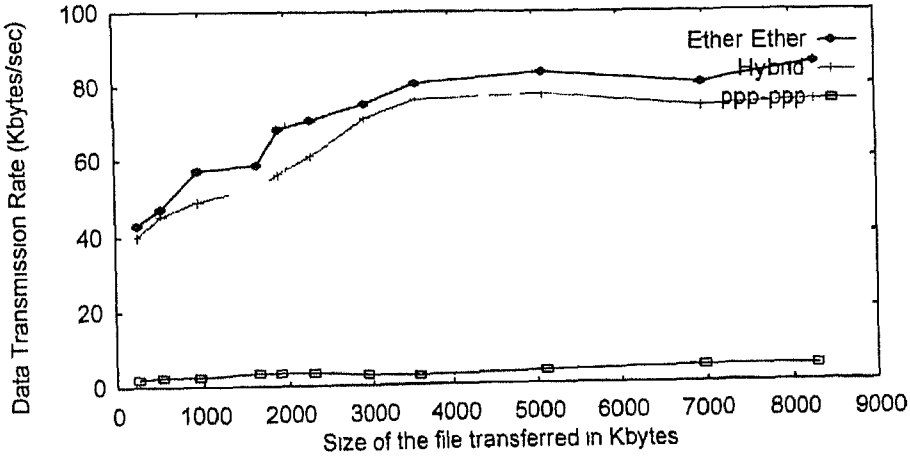


Figure 4 2 Comparison of Data Rates

The comparision of the Data Transmission Rates for the Asymmetric Internet aceess with respect to the ordinary access is shown in the Figure4 2 These are the observations taken from our Test Setup The ordinary access is done either by connecting the internet via the the telephone cables or via the fast ethernet When the connection is via the telephone lines a data rate of 3 5 Kbytes/sec is observed In both

the hybrid access and the ethernet access a data rate of 75 Kbytes/sec is observed for moderately large files (of the order of mega bytes) For smaller files a data rate of 45 Kbytes/sec is observed

The Data Transmission Rate is calculated from the data available from the Squid log file (access log) The file gives the total amount of data that is transferred for a connection and the amount of time taken This time is the sum of the time taken for the data to transfer from the application server to the client machine (via squid) and the time for the initial establishment of the connection If the two times are x sec and y sec respectively then the access log file gives $(x+y)$ sec

The above graph shows the variation of the transfer rates with respect to the size of the file For nearly all file sizes the x component of time will be present and it will depend on how fast the connection is established For small size files the x component dominates and hence the calculated transfer rates are low When the size of the file is large the y component dominates and more accurate data rates will be observed

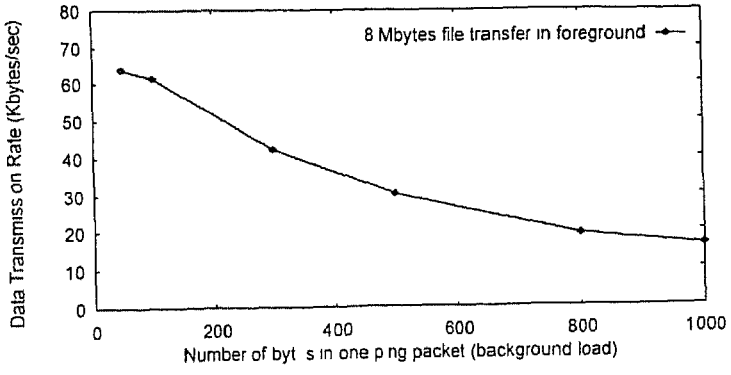


Figure 4 3 Effect of back ground traffic

The transmission rate is measured by gradually increasing the background traffic (the size of the ping packets is varied from 50 bytes to 1000 bytes) The variation of transmission rate is with respect to the background traffic as shown in Figure 4 3 When the low bandwidth telephone line is loaded with ping packets of 50 bytes the tiansmision rate of 65 Kbytes/sec was observed A rate of 16 5 Kbytes/sec was observed when loaded with 1000 byte ping packets

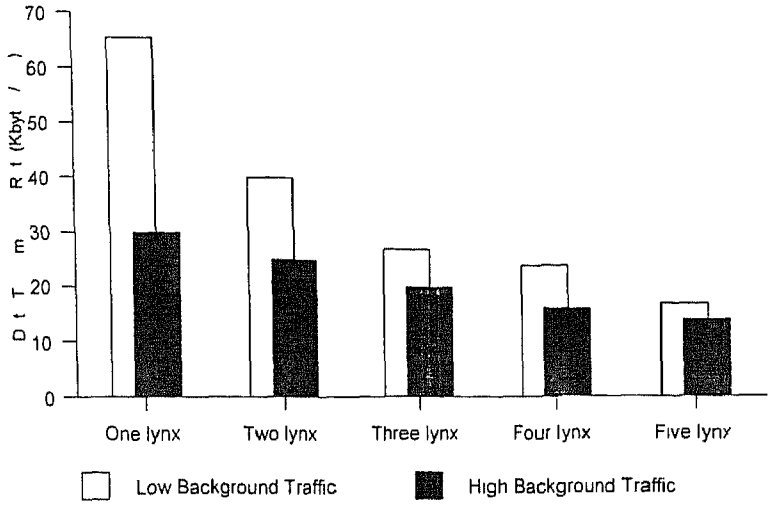


Figure 4 4 Data Transmission Rate vs Number of Connections

The above graph shows the variation of the transmission rate with the number of simultaneous connections Each connection is a ftp connection transferring a very large file (of the order of Mega bytes) The variation in transfer rate is shown both for low background traffic and high background traffic

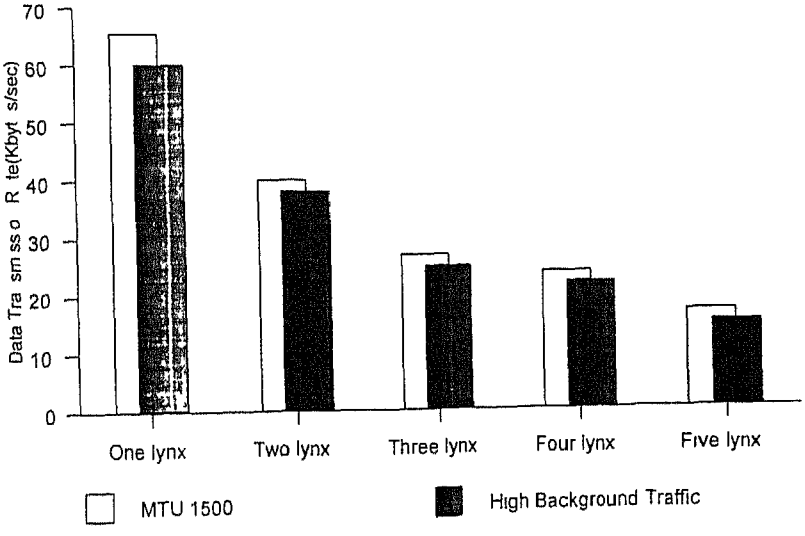


Figure 4 5 Data Transmission Rate vs MTU of High Bandwidth Sub network

The MTU of the interface which connects to the high bandwidth sub network (at hybrid gateway server) is varied and the data rates are measured The variation of the data transfer rates with respect to number of simultaneous connections is plotted in Figure 5 4 both for the MTU 1500 and MTU 576 When the MTU is less the TCP on the hybrid gateway server will generate more packets and hence more acknowledgements This will load the low bandwidth network This causes the slight fall in data rate as observed in Figure 4 5

Chapter 5

Conclusions

In this thesis we have described the implementation of a special asymmetric router for use in hybrid networks The hybrid network archicture is studied and the performance bottle necks and its solutions are described We have used the Linux Operation System for the implementation of the asymmetric router for the hybrid network The internals of the Linux kernel networking software are studied in depth so that we can properly enhance its functionality for the asymmetric data transfer required in such a network We have used the masquerading functionality of the Linux Operating System at the asymmetric router for connecting a private network to the internet The asymmetic router merges a low bandwidth telephone channel and a high bandwidth channel to provide large download rates for the users An algorithm is developed for reducing the congestion on the low bandwidth channel In case the return path introduces a large delay interactive connections such as telnet will become unattractive for the hybrid netwok To overcome this problem we have made the asymmetric access selective The system administrator at the asymmetric router can set rules to specify which services need hybrid transmission

For testing the performation of the hybrid network a test bed has been setup We used the telephone path as the forward path and the 10Mbps ethernet cable as the return path We observed nearly equal data rates(about 70Kbytes/sec) in both hybrid transmission and in ordinary transmission (with ethenet in both directions)

Suggestions for future work

The algorithm for doing the acknowledgment dropping may be studied further The algorithm basically drops any pending acknowledgments by looking at a queue The dropping of the acknowledgments may also be done based on other criteria such as delay (as observed for a particular connection) and the network state (i e start of congestion end of congestion and the network state) This hybrid network can be further extended to include an IP multicast feature

Bibiliography

[1] Vivek Arora Narin Suphasindhu and John S Baras *Asymmetic Internet Access over Satellite Terrestrial Networks*

[2] Vivek Shrama Class based Asymmetric Hybrid Server M Tech Thesis IIT Kanpur (in preparation)

[3] Y Rekhter R Moskowitz D Karrenberg *Address allocation for Private Internets* Internet Requests for Comments no 1918

[4] W R Stevens *TCP/IP Illustrated* Vol I Addison Wesley 1994

[5] D E Comer *Internetworking with TCP/IP* Vol I Principles Protocols and Architecture 2nd ed Prentice Hall 1991

[6] J Postel *Internet Protocol (IP)* Internet Requests for Comments no 791 770

[7] J Postel *Transmission Control Protocol (TCP)* Internet Requests for Comments no 793 761 675

[8] W Simpson *The Point to Point Protocol (PPP)* Internet Requests for Comments no 1661

[9] Olaf Kirch *The Linux Network Administrators Guide*

[10] Micheal K Johnson *Linux Kernel Hackers Guide*